Contextual Parameter Generation for Knowledge Graph Link Prediction

George Stoica* **Otilia Stretcu*** **Emmanouil Antonios Platanios***

Tom M. Mitchell **Barnabas Poczos**
Machine Learning Department, Carnegie Mellon University
`{gis, ostretcu,e.a.platanios,tom.mitchell,bapoczos}@cs.cmu.edu`

Abstract

We consider the task of knowledge graph link prediction. Recent approaches tackle this problem by learning entity and relation embeddings. However, they often constrain the relationship between these embeddings to be additive (i.e., the embeddings are concatenated and then processed by a sequence of linear functions and element-wise non-linearities). We show that this type of interaction significantly limits representational power, and instead propose to use *contextual parameter generation* to address this limitation. More specifically, we treat relations as the *context* in which entities are processed to produce predictions, by using relation embeddings to generate the parameters of a model operating over entity embeddings. We apply our method on two existing link prediction methods, including the current state-of-the-art, resulting in significant performance gains and establishing a *new state-of-the-art* for this task. These gains are achieved while also *reducing training time by up to 28 times*.

1 Introduction

Many real-world applications, from search engines to conversational agents such as Amazon's Alexa and Apple's Siri, rely on the ability to infer new facts from existing knowledge. A common means of representing such knowledge is via *knowledge graphs (KGs)*, where facts are represented as triples (e_s, r, e_t), and encode factual relationships between graph nodes. For each triple, we refer to e_s and e_t as the source and target entities, respectively, and we refer to r as the relation between e_s and e_t. While there exist many KGs that capture a diverse field of information, they are often incomplete [21]. However, many missing links are inferrable from existing knowledge. This motivates the task of *link prediction*, which is typically formulated as either question answering—inferring answers to questions of the form $(e_s, r, ?)$—or fact checking—evaluating the validity for statements of the form (e_s, r, e_t). This work focuses explicitly on the question answering formulation.

The study of link prediction has gathered substantial attention in the past years, and many methods have been proposed to solve it. These can be categorized as either single-hop or multi-hop models. Given a question, single-hop approaches [1, 23, 20, 14, 8] infer the answer directly, while multi-hop methods [11, 4, 15, 5, 19, 2, 13, 24, 6] find a path along the KG from the source entity to the most probable target entity. A significant boost in performance was observed when recent methods such as ConvE [3], MINERVA [2], or MultiHop-KG [13] combined KGs with the expressive power of neural networks. Such approaches learn finite dimensional continuous vector representations (i.e., embeddings) for both the entities and relations in the KG, and process them (e.g., through a neural network) to infer its missing links. However, in these models, interactions between entity and relation representations are restricted to being additive (e.g. they may be concatenated and linearly projected). In this work, we show how this type of interaction between entities and relations significantly limits expressive power, and we propose a novel method to address this limitation. More specifically, we

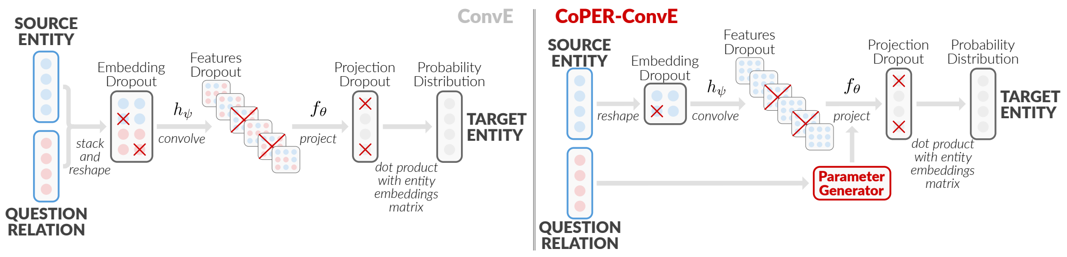

Figure 1: Architectures for ConvE (left) and its CoPER-enhanced version, CoPER-ConvE (right).

propose to treat the relations as the *context* in which source entities are interpreted and transformed to produce target entities. Concretely, we use the relation embeddings to generate the parameters of a model operating over entity embeddings, which then outputs a distribution over correct answers. The proposed method, **CoPER** (***Co**ntextual **P**arameters from **E**mbedded **R**elations*), has the following desirable properties: (1) **Abstract:** It can be used to enhance the representational power of several link prediction methods; (2) **Simple:** It can be formulated as a simple transformation for qualifying models, that can be implemented with only about 10 lines of code; (3) **Scalable:** It speeds up training by up to 28×; (4) **State-of-the-Art:** It outperforms competing methods by a significant margin on several established datasets.

2 Background

Notation. Let e_s, r, and e_t denote one-hot encoded representations of the source entity, relation, and target entity of a KG triple. A common approach to learning abstract representations of entities and relations is to learn vector embeddings, which provides for a simple yet effective method of sharing information. The transformation from one-hot encodings to vector embeddings is modeled as follows. Let N_e and N_r denote the total number of distinct entities and relations in the KG, respectively. Given a set of entities, $E = \{e_i\}_{i=1}^{N_e}$, and a set of relations $R = \{r_i\}_{i=1}^{N_r}$, we define the following embedding matrices: $\mathbf{E} \in \mathbb{R}^{D_e \times N_e}$ and $\mathbf{R} \in \mathbb{R}^{D_r \times N_r}$, where D_e and D_r correspond to the entity and relation embedding sizes, respectively. $\mathbf{E}$ and $\mathbf{R}$ are both trainable parameters. Given a question of the form $(e_s, r, ?)$, the corresponding source entity and relation embeddings are $\mathbf{e}_s = \mathbf{E}e_s$ and $\mathbf{r} = \mathbf{R}r$, where $\mathbf{e}_s \in \mathbb{R}^{D_e}$ and $\mathbf{r} \in \mathbb{R}^{D_r}$, respectively.

ConvE. ConvE [3] is the current state-of-the-art and one of the baselines used in our experiments. In ConvE, the target entity is estimated through the function: $\mathbf{e}_t = \text{Dense}_\phi(\text{Conv2D}(\text{Reshape}([\mathbf{e}_s; \mathbf{r}])))$. Here, $[\mathbf{e}_s; \mathbf{r}] \in \mathbb{R}^{D_e + D_r}$ represents the result of stacking the entity and relation embeddings together, followed by a reshape into a $W \times H$ rectangular matrix, where W and H are model hyperparameters such that $WH = D_e + D_r$. This matrix is then passed through a 2D convolution layer, followed by a dense layer, to obtain the answer entity embedding $\mathbf{e}_s$. The dense layer is defined as a ReLU activated linear-layer, where ϕ denotes its weight matrix and bias vector combined. An illustration of the ConvE model is shown in the left of Figure 1. For further details, we refer the reader to the work of [3]. Importantly, we observe that the entity and relation embeddings are concatenated as input to the convolution, which induces an *additive* interaction between them. As we introduce in the following paragraph, this induces a restriction on the expressibility of the model.

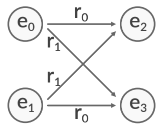

Figure 2: Toy example.

Limited Expressive Power. Many existing neural methods consist of the following steps: (i) learn entity and relation embeddings, (ii) concatenate the source entity and relation embeddings, and (iii) perform a sequence of linear transformations and element-wise non-linear operations on the concatenated vector. Through these types of operations, the elements of the entity embedding, $\mathbf{e}_s$, and relation embedding, $\mathbf{r}$, will only interact in an *additive* way. To understand what we mean by *additive*, consider the case where $\mathbf{e}_s$ and $\mathbf{r}$ are concatenated and then projected using the weight matrix $\mathbf{W}$. The output is given by $\mathbf{W}[\mathbf{e}_s; \mathbf{r}]$, which is equivalent to $\mathbf{W}_e\mathbf{e}_s + \mathbf{W}_r\mathbf{r}$, where $[\mathbf{W}_e; \mathbf{W}_r]^T = \mathbf{W}$. Note that no elements of $\mathbf{e}_s$ are multiplied with any from $\mathbf{r}$. This imposes a restriction on what types of KG structures can be modeled explicitly, and even the simple example in Figure 2 poses an issue. Due to space limitations, we leave the proof in Appendix A, and we show how our proposed method addresses this limitation in Appendix B. However, it suffices to note that our proposed approach alleviates the constraints imposed by these additive interactions.

3 Model

Our method, termed **CoPER** — *Contextual Parameters from Embedded Relations*, can be used to enhance multiple existing additive link prediction methods, by enabling them to learn more expressive relationships between entities and relations. At the core of CoPER lies the key idea that relations define *how* source entities are processed in order to produce answer entities (i.e. relations are the *context* in which we process the entities). Specifically, when answering a question $(e_s, r, ?)$, the target entity e_t can be obtained through a transformation of the source entity e_s, and the *parameters* of this transformation are determined by the relation r. In this section, we propose and compare different potential architectures for this contextual parameter generation (**CPG**) module.

Parameter Generator Network. Let h_ϕ denote an arbitrary merge function, with parameters ϕ, that additively combines e_s and r. In the example above, $h_\phi(e_s, r) = \mathbf{W}[\mathbf{e}_s; \mathbf{r}]$ with $\phi = \mathbf{W}$. In CoPER, we introduce a function that takes as input r and outputs the parameters ϕ of h, while h now only operates over e_s. Let $g\colon \{1, 2, ..., N_r\} \to \mathbb{R}^{D_\phi}$ be our parameter generation function, where N_r is the number of relations in the KG, $\phi \in \mathbb{R}^{D_\phi}$, and D_ϕ is the number of parameters in ϕ. We now present three simple functional forms for g that we also use for our experiments.

Parameter Lookup Table. The simplest approach is to output an entirely different ϕ for each relation. This results in the following form: $g_{\text{lookup}}(r) = \mathbf{W}_{\text{lookup}} r$, where r here is a one-hot encoded vector representation of the relation, and $\mathbf{W}_{\text{lookup}} \in \mathbb{R}^{D_W \times N_r}$ is the only learnable parameter of g_{lookup}, with $D_\phi = D_W \cdot N_r$. However, the problem with this simple formulation is that information sharing across relations can only happen through the shared entity embeddings. As we illustrate in Appendix F, this makes the model prone to overfitting. Moreover, in large KGs, many of the relations may be similar (e.g., `bornIn` and `livesIn`), and it may be beneficial for these relations to share information. This motivates a different approach for generating parameters.

Linear Projection. Instead of using one-hot encodings for the relations, we can learn embeddings: $g_{\text{linear}}(r) = \mathbf{W}_{\text{linear}} \mathbf{R} r + \mathbf{b}$, where we use the embedding lookup equation, $\mathbf{W}_{\text{linear}} \in \mathbb{R}^{D_W \times D_r}$, bias term $\mathbf{b} \in \mathbb{R}^{D_W}$, D_r is the relation embedding size, and $\mathbf{W}_{\text{linear}}$, $\mathbf{b}$, and $\mathbf{R}$ are trainable model parameters. Thus, $\phi = (\mathbf{W}, \mathbf{b})$, with $D_\phi = D_W \cdot (D_r + 1)$. Intuitively, the learned relation embeddings represent a linear combination (offset by $\mathbf{b}$) of D_r different values for ϕ, allowing for shared information between relations.

Multi-Layer Perceptron. Most of our experiments are performed using g_{linear}, with which we achieve state-of-the-art results. However, we observed that g_{linear} underperforms for small datasets. We believe that this is most likely due to $\mathbf{W}_{\text{linear}}$ becoming too big relative to the original number of parameters. This is because, if our original additive network had $D_\phi = D_W$ trainable variables, g_{linear} introduces $D_W \times (D_r + 1)$ parameters, which makes g_{linear} significantly larger. Limiting the value of D_r is not necessarily a solution as a small D_r can significantly constrain the capacity of our model. We therefore propose a third variant of the generator network using a multi-layer perceptron: $g_{\text{MLP}}(r) = \text{MLP}(\mathbf{R} r)$. This can be thought of as a low-rank approximation to g_{linear}.

These are only three possible proposals for the parameter generator network and, as we show in Appendix B, even a simple network such as g_{linear} already significantly increases representational power. Note however, that the idea behind CoPER is more general and can be extended to more complex architectures. In contrast to similar work discussed in Appendix H, CPG learns more powerful relationships between relations; the network can learn any arbitrary relation interactions. Furthermore, in comparison to these methods (refer to Appendix H), CoPER achieves the best performance.

CoPER-ConvE. In CoPER-ConvE, the first pre-processing steps in the pipeline (reshape, convolution) are only applied to the entity embedding, while the relation is now used to generate the parameters of the projection layer:

$$\mathbf{z} = \text{Conv2D}(\text{Reshape}(\mathbf{e}_s)), \quad \phi = g(r), \quad \text{and} \quad \hat{e}_t = \text{Dense}_\phi(\mathbf{z}) = \phi_1 + \phi_{2:D_\phi} \mathbf{z},$$

where ϕ is the parameter vector produced by the Parameter Generator, with its first element ϕ_1 used as bias in the projection layer, and the other $D_\phi - 1$ elements being used as a weight matrix.

CoPER-MINERVA. MINERVA is a multi-hop method, which means that it will answer the question $(e_s, r, ?)$ by finding a path in the graph that connects e_s with the predicted answer $\hat{e}_t$. At each hop in the graph, the model is repeatedly invoked to predict the next target entity to move to. Due to space constraints, we describe this architecture in more detail in Appendix C. However, what is important to note is that MINERVA's architecture contains two points where entity and relation embeddings are

Table 1: Results for various link prediction models. Results for ConvE, MINERVA, CoPER-ConvE and CoPER-MINERVA are reported according to our own experiments. The remainder are taken from [2]. All numbers are expressed as percentages. "*" denotes experiments in progress, while "-" denotes missing results from the respective publications.

Dataset	**Metric**	**Models**								
		DistMult	**ComplEx**	**NeuralLP**	**NTP-λ**	**MINERVA**	**MultiHop-KG**	**ConvE**	**CoPER-MINERVA**	**CoPER-ConvE**
	`Hits@1`	82.1	82.3	64.3	84.3	75.3	90.2	92.89	77.76	**95.46**
UMLS	`Hits@10`	96.7	99.5	96.2	**100.0**	96.7	99.2	99.70	97.43	99.70
	`MRR`	86.8	89.4	77.8	91.2	84.1	94.0	95.35	85.44	**97.08**
	`Hits@1`	48.7	75.4	47.5	75.9	60.5	78.9	74.21	66.20	**83.62**
Kinship	`Hits@10`	90.4	98.0	91.2	87.8	92.4	98.2	97.86	94.23	**98.42**
	`MRR`	61.4	83.8	61.9	79.3	72.0	86.5	83.04	76.00	**89.52**
	`Hits@1`	43.1	41.0	37.6	–	41.3	41.8	41.86	42.66	**44.05**
WN18RR	`Hits@10`	52.4	51.0	**65.7**	–	51.3	51.7	52.17	50.99	56.12
	`MRR`	46.2	44.0	46.3	–	44.8	45.0	45.19	46.51	**48.33**
	`Hits@1`	32.4	15.8	16.6	–	22.3	**32.7**	30.30	29.49	32.18
FB15k237	`Hits@10`	60.0	42.8	34.8	–	44.9	56.4	60.83	50.39	**62.92**
	`MRR`	41.7	24.7	22.7	–	29.2	40.7	40.51	36.51	**42.56**
	`Hits@1`	55.2	64.3	–	–	66.3	65.6	67.04	*	**70.25**
NELL-995	`Hits@10`	78.3	86.0	–	–	83.1	84.4	87.96	*	**88.70**
	`MRR`	64.1	72.6	–	–	72.5	72.7	75.42	*	**77.48**

combined, and in both cases the combination is additive. With CoPER we replace these merge steps with a CPG module.

4 Experiments

Datasets. We adopt the following datasets used in prior literature: Unified Medical Language Systems (*UMLS*) [10], Alyawarra *Kinship*, *WN18RR* [3], *FB15k-237*[18], and NELL-995 [22]. Table 2 in Appendix D shows summary statistics for each dataset.

Models. We evaluate CoPER-ConvE and CoPER-MINERVA against their base models and multiple other link prediction methods. Regarding the parameter generation module, we perform experiments using both g_{linear} and g_{MLP}. Further details with respect to hyperparameters, batch size and optimizier can be found in Appendix E.

Results. We report results for two metrics used throughout prior work: `Hits@k` and Mean Reciprocal Rank (`MRR`). Further details about what these represent can be found in Appendix E. Our overall performance results are reported in Table 1. We observe that CoPER-ConvE outperforms ConvE on all datasets, with up to $+9.41\%$ `Hits@1` performance gain over ConvE on Kinship. Moreover, we find that CoPER-ConvE achieves better performance over all other existing methods on these datasets, often by a significant margin. Notably, we observe a +4.7% `Hits@1` gain for Kinship over the best existing method and a +3.21% `Hits@1` gain for NELL-995. To the best of our knowledge, CoPER-ConvE establishes a *new state-of-the-art* on all these datasets.

We also examine the effect of CoPER on *training time*. Since CoPER-ConvE is the version with the best results across all datasets, we perform this analysis for ConvE and CoPER-ConvE. Given that CoPER-ConvE consistently outperforms ConvE in terms of `Hits@1` we compare the number of iterations that each method requires to reach the best `Hits@1` value that ConvE achieves (e.g., we check when both ConvE and CoPER reach 92.89% `Hits@1` on UMLS). Then we calculate the ratio: $\frac{\text{\# iterations CoPER}}{\text{\# iterations ConvE}}$. For instance, if a baseline model requires 10,000 steps to attain best performance, while its CoPER variant takes 3,000 steps to achieve identical performance, then this metric would be: $\frac{3{,}000}{10{,}000} = 0.3$. Correspondingly, the training speedup would be $\frac{1}{0.3} = 3.33$. Our results, illustrated in Figure 4 in Appendix G, show that CoPER-ConvE always requires much fewer training iterations than ConvE, yielding a speedup between $2.9\times$ to $28.6\times$.

5 Conclusion

We propose CoPER, a novel framework that improves upon the current state-of-the-art methods for the task of knowledge graph link prediction. CoPER treats relations as the context in which source entities are processed to predict target entities. We show how this significantly increases the expressive power of link prediction models by allowing them to represent multiplicative interactions between entities and relations. We also show our approach's flexibility by extending both a single-hop and a multi-hop link prediction model, achieving new state-of-the-art performance for this task, while significantly speeding up training time over unaltered methods by up to $28\times$.

References

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[2] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[3] Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pages 1811–1818, February 2018.

[4] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. 2013.

[5] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *EMNLP*, 2015.

[6] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. *CoRR*, abs/1506.01094, 2015.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[8] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, 2015.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[10] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, pages 433–440. ACM, 2007.

[11] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011.

[12] Ben Lengerich, Andrew Maas, and Christopher Potts. Retrofitting Distributional Embeddings to Knowledge Graphs with Functional Relations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2423–2436, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[13] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253, 2018.

[14] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2181–2187. AAAI Press, 2015.

[15] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *ACL*, 2015.

[16] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. Contextual Parameter Generation for Universal Neural Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium, November 2018.

[17] Sashank Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*, 2018.

[18] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. 2015.

[19] Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*, 2016.

[20] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, volume 48, pages 2071–2080, 2016.

[21] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge Base Completion via Search-Based Question Answering. In *WWW*, 2014.

[22] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573. Association for Computational Linguistics, 2017.

[23] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015.

[24] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base completion. *CoRR*, abs/1702.08367, 2017.

Appendices

A Limitations of Additive Interactions

Most existing neural methods consist of the following steps: (i) learn entity and relation embeddings, (ii) concatenate the source entity and relation embeddings, and (iii) perform a sequence of linear transformations and element-wise non-linear operations on them, in order to obtain the target entities. We now use an example to explain why this only explicitly allows for additive interactions between the source entity and relation, and why this is significantly limiting the model's expressive power. Consider a simple merging function where the source entity and relation are first concatenated into a single vector as $[\mathbf{e}_s; \mathbf{r}]$, and then projected through a linear layer:

$$h_\phi(\mathbf{e}_s, \mathbf{r}) = \phi \cdot [\mathbf{e}_s; \mathbf{r}], \tag{1}$$

where $\phi \in \mathbb{R}^{D_z \times (D_e + D_r)}$ and D_z is the size of $\mathbf{z}$. If we refer to the first D_e columns of ϕ as ϕ_e, and the last D_r columns as ϕ_r (i.e., $\phi = [\phi_e; \phi_r]$), then we can write $h_\phi(\mathbf{e}_s, \mathbf{r}) = \phi_s \mathbf{e}_s + \phi_r \mathbf{r}$. Note that, in this case the elements of the entity embedding $\mathbf{e}_s$ and the relation embedding $\mathbf{r}$ only interact in an additive way (i.e., the output $\mathbf{z}$ is a linear combination of the elements in $\mathbf{e}_s$ and $\mathbf{r}$ and it does not support more complex interactions, such as multiplicative or polynomial). The same is true for the merging function in ConvE through Conv2D (only some of the elements of ϕ are shared), as well as the merging functions of MINERVA and Multihop-KG. The main implication of this is that relations cannot influence the projection matrices used to transform the entities.

Let us demonstrate this important limitation by considering the example shown in Figure 2, illustrating 4 KG facts: $(\mathtt{e}_0, \mathtt{r}_0, \mathtt{e}_2)$, $(\mathtt{e}_0, \mathtt{r}_1, \mathtt{e}_3)$, $(\mathtt{e}_1, \mathtt{r}_0, \mathtt{e}_3)$, $(\mathtt{e}_1, \mathtt{r}_1, \mathtt{e}_2)$. Suppose we want to encode these facts using a model in the form of Equation 1:

$$\mathbf{e}_2 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_0 \tag{2}$$
$$\mathbf{e}_3 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_1 \tag{3}$$
$$\mathbf{e}_3 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_0 \tag{4}$$
$$\mathbf{e}_2 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_1 \tag{5}$$

Subtracting (3) from (2), and (5) from (4), we have that:

$$(\mathbf{e}_2 - \mathbf{e}_3) = \phi_r(\mathbf{r}_0 - \mathbf{r}_1)$$
$$(\mathbf{e}_3 - \mathbf{e}_2) = \phi_r(\mathbf{r}_0 - \mathbf{r}_1)$$

which leads to a degenerate solution where $\phi_r = 0$, $\mathbf{r}_0 = \mathbf{r}_1$, or $\mathbf{e}_3 = \mathbf{e}_2$. This implies that additive models cannot represent this toy example.

While this is a toy example, it illustrates a more general problem. For instance, consider a case where we want to learn a different expert model for each relation. This means that given a source entity and relation, the relation determines which expert to use when processing the source entity. This example is important because related work in other areas has shown that mixtures of experts—our toy example is in fact a very simple form of a mixture of experts—can result in significant performance gains [12]. Methods that combine entities and relations additively cannot learn such a mixtures of experts. Ideally, we want our model to be expressive enough such that it can learn functions that are conditional on the relation (such as the above mixture of experts example). The latter case is important because learning a separate model for each relation may sometimes be impossible. A common example for large KGs is that for some relations we have a lot of training data, but for most we have very little. In such cases, we want to be able to leverage the fact that many relations are similar by sharing information among them. To this end, we propose to use *contextual parameter generation*, originally proposed in the context of neural machine translation by [16]. Importantly, as we show in Appendix B the proposed approach is able to handle the aforementioned toy example as well as more general mixtures of experts.

B Enhanced Expressive Power with Contextual Parameter Generation

Through the parameter generation component explained in Section 3, CoPER enables link prediction models to learn functions that depend on more complex interactions between the entity and

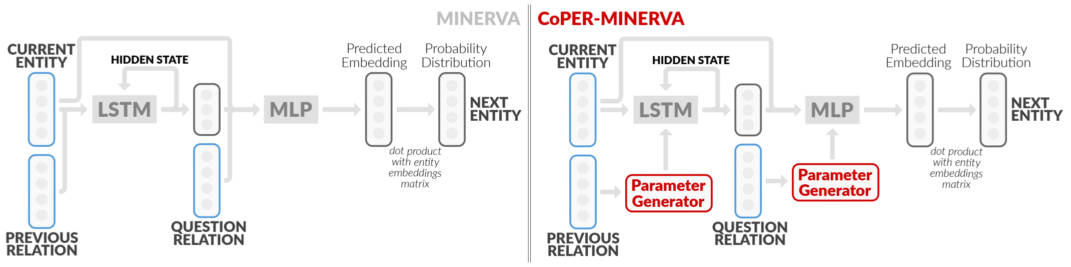

Figure 3: MINERVA versus CoPER MINERVA.

relation embeddings. A CPG module as simple as g_{linear}, combined with any typical neural network architecture for h_ϕ (from a single linear layer to many complex layers followed by element-wise non-linearities) allows the model to represent *multiplicative* interactions between source entities and relations.

For ease of explanation, we will illustrate this increase in representation power with a simple form of h_ϕ, but it is easy to see that more complex architectures can only increase the expressive power further. According to the CoPER formulation, h_ϕ now only operates on $\mathbf{e}_s$. For simplicity, let $h_\phi(\mathbf{e}_s) = \phi\mathbf{e}_s$. The parameters ϕ are given by $\phi = g_{\text{linear}}(r) = \mathbf{WR}r + \mathbf{b} = \mathbf{Wr} + \mathbf{b}$. Thus, we have that:

$$\hat{e}_t = h_\phi(\mathbf{e}_s) = h_\phi(\mathbf{e}_s) = \phi\mathbf{e}_s = (\mathbf{Wr} + \mathbf{b})\mathbf{e_s}. \tag{6}$$

This result shows that the relation and entity embedding now interact in a multiplicative way, which means the relation itself affects the weights with which we multiply the entity embedding. This is more expressive than an additive interaction, as it now allows us to represent dependencies such as conditionals (i.e., if statements), mixtures of experts, and even the toy example we in Figure 2.

Toy Example. Going back to our toy example that additive interactions cannot represent, we now show that a CPG module as simple as g_{linear} or g_{lookup} can encode this KG example. Applying the predictor derived in Equation 6 to the KG in Figure 2, the following equations must hold for the toy example to be representable by the model:

$$\mathbf{e}_2 = (\mathbf{Wr}_0 + \mathbf{b})\mathbf{e}_0 \tag{7}$$
$$\mathbf{e}_3 = (\mathbf{Wr}_0 + \mathbf{b})\mathbf{e}_1 \tag{8}$$
$$\mathbf{e}_2 = (\mathbf{Wr}_1 + \mathbf{b})\mathbf{e}_1 \tag{9}$$
$$\mathbf{e}_3 = (\mathbf{Wr}_1 + \mathbf{b})\mathbf{e}_0 \tag{10}$$

Subtracting (7) from (8), and (9) from (10), we have that:

$$\mathbf{e}_3 - \mathbf{e}_2 = (\mathbf{Wr}_0 + \mathbf{b})(\mathbf{e}_1 - \mathbf{e}_0)$$
$$\mathbf{e}_3 - \mathbf{e}_2 = (\mathbf{Wr}_1 + \mathbf{b})(\mathbf{e}_0 - \mathbf{e}_1)$$

which, avoiding the degenerate solution where $\mathbf{e}_0 = \mathbf{e}_1$, leads to:

$$\mathbf{W}(\mathbf{r}_0 - \mathbf{r}_1) + 2\mathbf{b} = 0$$

Since this equation has an infinite number of solutions, any $\mathbf{W}$ and $\mathbf{b}$ that satisfy this condition can represent the subgraph in Figure 2.

Note that although we showed here that CPG leads to multiplicative interactions between $\mathbf{e}_s$ and $\mathbf{r}$ for a particular choice of $h_\phi(x) = \phi x$, the conclusions will stand for most neural network architectures, from multilayer percepron to convolutional to recurrent neural networks, since they usually involve such a projection step on the inputs.

C MINERVA.

MINERVA [2] is a deterministic RL-based multi-hop question-answering agent. The model defines states as the entities in the KG, and actions as tuples (r, e) consisting of an outgoing relation and its destination entity, specifying a hop to a neighboring node in the KG. Given a question (e_s, r_q, e_t), MINERVA traverses the KG along its relations from e_s to the most likely target entity e_t. Each step along the graph path iteratively accumulates a history of entities and relations visited, which is aggregated together through and then stored in a Long Short-Term Memory (LSTM) network [7], as illustrated in Figure 3 (left). The hidden state of the LSTM is updated as follows:

$$\mathbf{h}_i = \text{LSTM}(\mathbf{h}_{i-1}, [\mathbf{e}_i; \mathbf{r}_{i-1}]), \qquad \text{(merge)}$$

where $\mathbf{h}_i$ denotes the accumulated history representation at the i^{th} time step, $\mathbf{h}_{i-1}$ is the history representation at the previous step and is the hidden state of the LSTM, $\mathbf{r}_{i-1}$ denotes the embedding representation of the relation taken in the previous step leading to state e_i (represented by the embedding $\mathbf{e}_i$), and $[;]$ represents vector concatenation. Additionally, $[\mathbf{e}_i; \mathbf{r}_{i-1}]$ denotes the LSTM input. Because the LSTM module consists of a series of input projections, $\mathbf{e}_i$ and $\mathbf{r}_{i-1}$ are *additively* incorporated into the agent's history.

At every time step, once the traversal history has been accumulated, the agent next determines the subsequent action to take as follows:

$$o_i = \text{MLP}([\mathbf{h}_i; \mathbf{e}_i; \mathbf{r}_q]), \qquad \text{(merge)} \qquad (11)$$

$$a_j = \text{Categorical}(A_i o_i), \qquad \text{(prediction)} \qquad (12)$$

where A_i denotes the embedding representations of each available action from e_i, o_i represents the Multi-Layer Perception (MLP) output, Categorical denotes a categorical distribution decision function—such as a network policy—which operates over action distribution logits given by $A_i o_i$, and a_j is the selected action. Since an action is a tuple (r, e) as explained above, we represent a_j as the concatenation of the respective relation embedding and an entity embedding. A_i denotes then the matrix containing vector representations of all available actions from each state.

Importantly, we observe that the entity and relation embeddings are concatenated as input to the MLP, which also induces an *additive* interaction between them in this component. Thus, in both components where entity and relation information is processed in MINERVA, it is done additively. As illustrated with our toy example, this limits the impressibility of the network.

C.1 CoPER-MINERVA.

As mentioned above, MINERVA contains two points in the model where a relation embedding is concatenated with another vector containing entity information. We replace this step in both cases with a parameter generator, as illustrated in the right of Figure 3. In the fist case, the embedding of the previous relation in a step, $\mathbf{r}_{i-1}$, is used as input to a parameter generator that outputs the parameters of the LSTM component. In the second case, the query relation $\mathbf{r}$ embedding is used to generate the parameters of the MLP, which operates over the step history and current entity representations. The rest of the model remains unchanged.

D Dataset Statistics.

Table 2 displays summary statistics of the benchmark datasets used in our evaluation.

Table 2: Dataset statistics. Here, # Train denotes the number of questions used for training, N_e the number of distinct entities, N_r the number of distinct relations, $\bar{N}_a$ the average number of answers per question, and $\bar{d}$ the average degree of the graph nodes in the dataset.

Dataset	# Train	N_e	N_r	$\bar{N}_a$	$\bar{d}$
Kinship	8,544	104	25	6.14	82.15
UMLS	5,216	135	46	7.83	26.59
FB15k237	272,115	14,541	237	3.03	17.87
WN18RR	86,835	40,945	11	1.41	2.19

E Further Experimental Details

We use the evaluation strategies presented in both ConvE and MINERVA for CoPER-ConvE and CoPER-MINERVA, respectively, to generate a ranking over entities at evaluation time. For instance, ConvE obtains a distribution over answer entities by computing the dot product between the predicted entity target embedding and the embeddings of all possible entities in the KG, and then ranks them [3]. Conversely, MINERVA obtains a probability distribution over target entities by performing a beam search and ranking all possible target entities by their path probabilities [2].

Metrics. We report results for two metrics used throughout prior work: `Hits@k` and Mean Reciprocal Rank (`MRR`). Both assess how a model ranks the correct answer compared to all other possible answers. `Hits@k`, also known as recall-at-k, is defined as the proportion of times the correct answer is ranked among the top-*k* answers, according to the probabilities assigned by the model. Similar to prior work, we report the average `Hits@1` and `Hits@10` over the test set. `MRR` is defined as the average value of the reciprocated rank of the correct answer for each test instance. Therefore, `MRR` is a measure of the overall quality of a model's predictions. Note that this evaluation method is also used in MINERVA.

Training. All our experiments were performed on a machine with a single Nvidia Titan X GPU. We used the AMSGrad optimizer [17] for our CoPER-ConvE and ConvE experiments, and the Adam Optimizer [9] for our CoPER-MINERVA and MINERVA experiments. Accounting for GPU memory limits, our batch sizes were 512 and 128 respectively, and we used a learning rate of .001 across all experiments.

Hyperparameters. For CoPER, we vary the relation embedding size (originally 200) based on the number of relations in each dataset, which stems from our observations that datasets with few relations (e.g. Kinship or WN18RR) perform better with smaller embeddings. We choose the dropout parameters by performing a grid search between [0,1] based on the validation set `Hits@1`. For the g_{MLP}, we use a single hidden layer with a ReLU activation and chose the number of hidden units by also performing a grid search between. The exact hyperparameter values and CPG architecture utilized in CoPER can be found in our repository at `www.github.com/otiliastr/coper`.

F Ablation Tests

Table 3 shows results from our ablation experiments. Note that training CoPER-PL-ConvE on NELL-995 is infeasible with our resource capabilities due to the memory required to store the parameter lookup table and entity embeddings. While this illustrates an important disadvantage of CoPER-PL-ConvE, it also highlights CoPER-ConvE's capability to operate efficiently over large datasets.

Table 3: Overview of our ablation testing performances on our evaluation datasets. Results for ConvE, CoPER-PL-ConvE, and CoPER-ConvE are reported according to our own experiments. All numbers are expressed as percentages. "N/A" denotes experiments outside our resource capabilities.

Dataset	Metric	Models		
		ConvE	CoPER-PL-ConvE	CoPER-ConvE
UMLS	`Hits@1`	92.89	73.82	**95.46**
	`Hits@10`	**99.70**	99.09	**99.70**
	`MRR`	95.35	85.17	**97.08**
Kinship	`Hits@1`	74.21	74.90	**83.62**
	`Hits@10`	97.86	96.63	**98.42**
	`MRR`	83.04	83.22	**89.52**
WN18RR	`Hits@1`	41.86	**44.10**	44.05
	`Hits@10`	52.17	51.20	**56.12**
	`MRR`	45.19	46.63	**48.33**
FB15k237	`Hits@1`	30.30	30.72	**32.18**
	`Hits@10`	60.83	60.04	**62.92**
	`MRR`	40.51	40.52	**42.56**
NELL-995	`Hits@1`	67.04	N/A	**70.25**
	`Hits@10`	87.96	N/A	**88.70**
	`MRR`	75.42	N/A	**77.48**

G Training Speedup

Figure 4 illustrates the significant training time speedups CoPER-ConvE achieves over ConvE.

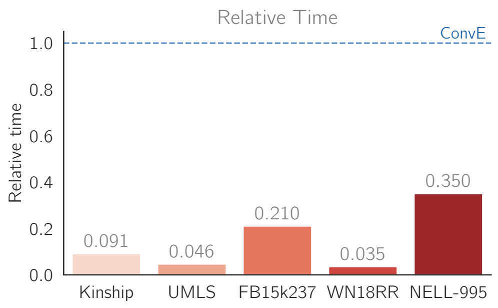

Figure 4: Time required for CoPER-ConvE to obtain its best performance on each dataset, as a fraction of the time it takes ConvE to achieve equivalent performance.

H Comparison with Prior Work

In this section we discuss CPG comparisons to prior work.

Interestingly, g_{lookup} is comparable to DistMult, TransR, and the composition network proposed by [6]. This is because each of these methods also use relations to define distinct mappings over entity embeddings. However, for single-hop methods: DistMult and TransR, sharing information across relations is constrained to just through the entity embeddings.

In contrast to CTransR [14] and TransD [8], which like g_{linear} and g_{MLP} also learn relationships between relations, CPG learns more expressive relationships between them. Based on the choice of CPG module, the network can learn any arbitrary relation interactions. Furthermore, its abstract design enables it to fully benefit from the expressive power of neural networks.

The results for TransR and TransD are as reported at `https://github.com/thunlp/OpenKE`, and all numbers represent `Hits@1`. We note that results for CTransR [14] are unavailable due to its unmaintained[1] or unavailable[2] implementation. While CoPER-MINERVA performs similarly to TransR, CoPER-ConvE significantly outperforms all other models on both datasets.

Table 4: Overview of `Hits@1` comparisons between CoPER models, TransR and TransD on FB15k237 and WN18RR.

Model	**Dataset**	
	WN18RR	**FB15k237**
TransR	51.9	51.1
TransD	50.8	48.7
CoPER-ConvE	**56.12**	**62.97**
CoPER-MINERVA	50.99	50.39
ConvE	52.27	60.83
MINERVA	51.3	56.4

[1]Its original implementation: `https://github.com/thunlp/KB2E` is no longer maintained and we were unable to train.

[2]It is missing from the official repository: `https://github.com/thunlp/OpenKE`.